2026

PENSEZ COMME UN PROGRAMMEUR

Les bases de la logique algorithmique expliquées simplement.

Ir. Alexandre ETOMASANGO OWANGA

Enseignant-Chercheur | Développeur Informatique

SOMMAIRE

INTRODUCTION 2
CHAPITRE 1 : LES ELEMENTS DE BASE D'UN ALGORITHME 4
1.1. QUELQUES DÉFINITIONS DE CONCEPTS 4
1.2. LES TYPES DE VARIABLES ET CONSTANTES 4
1.3. LES OPÉRATEURS 5
1.4. LES INSTRUCTIONS DE BASE 6
CHAPITRE 2 : L'ALGORIGRAMME & STRUCTURE DE CONTROLE 7
2.1. LES SYMBOLES 7
2.2. L'ALGORITHMIQUE AU QUOTIDIEN 10
2.3. LES STRUCTURES DE CONTRÔLE 11
CONCLUSION 13
CHAPITRE 3 : METHODOLOGIE D'ANALYSE D'UN PROBLEME 15
3.1. EXEMPLE 1 : LE CALCUL DE LA MOYENNE (CAS D'UN FICHIER EN ENTREE) 16
ALGORIGRAMME 17
3.3 EXEMPLE 2 (CAS DE LA SAISIE SUR CLAVIER) 18
CONCLUSION 19
SECTION EXERCICES : À VOUS DE JOUER ! 20
PARTIE 1 : Traitement à partir d'un fichier en entrée 20
PARTIE 2 : Traitement interactif avec saisie utilisateur 21

INTRODUCTION

Depuis quelques années, l'intelligence artificielle générative a bouleversé notre manière d'apprendre et de créer. Ce qui exigeait autrefois des mois d'études comme concevoir un site web ou un programme complexe peut désormais être généré en quelques secondes. Là où il fallait maîtriser le HTML, le CSS, le SQL ou le PHP, un étudiant peut aujourd'hui simplement dire : *« Génère-moi une application de gestion de stock avec une base de données »*, et obtenir un prototype fonctionnel. Cependant, cette facilité apparente cache un **danger invisible**.

L'IA n'est pas un cerveau, c'est un miroir. Elle ne réfléchit pas, elle prédit. Aujourd'hui, nous voyons apparaître une génération de **"bâtisseurs de sable"** c.à.d. des personnes capables de générer des logiciels qu'ils ne comprennent pas. Sans une maîtrise profonde de l'algorithmique, toute construction logicielle reste fragile, limitée et périlleuse.

Un logiciel de gestion (Desktop, Web ou Mobile) sans base logique solide finit toujours par s'écrouler : calculs financiers erronés, failles de sécurité, performances médiocres et incapacité totale de l'auteur à expliquer ou à faire évoluer son propre travail.

L'informatique de gestion repose sur un principe simple : **transformer des données brutes en informations stratégiques.** Derrière chaque tableau de bord, chaque rapport financier ou chaque indicateur de performance, il y a un moteur invisible : **l'algorithme**.

Un algorithme est une méthode organisée, logique et précise. Dans une entreprise, chaque opération peut être décomposée en une suite d'instructions :

- Comment calculer une facture sans erreur ?
- Comment calculer la moyenne semestrielle d'un étudiant sans erreur ?
- Comment automatiser les écritures comptables ?
- Comment prévenir une rupture de stock avant qu'elle n'arrive ?

La gestion exige une précision absolue. Une erreur de virgule ou une faille logique dans un calcul de salaire peut paralyser une organisation. C'est ici que l'algorithme intervient pour garantir la fiabilité, la cohérence et la transparence du système d'information.

Qu'on se le dise ! Un développeur qui ne maîtrise pas l'algorithmique est un passager de la technologie, pas un pilote. Il dépend entièrement des IA, des tutoriels YouTube et du code des autres.

À l'inverse, celui qui maîtrise l'algorithme possède la **compétence mère**. Il ne subit pas l'IA, il l'augmente. L'IA peut écrire le code, mais elle ne peut pas réfléchir à votre place. Elle ne comprendra jamais vos processus métiers, vos

contraintes de terrain ou la subtilité de votre organisation mieux que vous. Elle n'invente pas la logique : elle reçoit la vôtre.

Dans ce livre, nous revenons à l'essentiel. Nous allons apprendre à comprendre et à analyser un problème avant d'écrire un algorithme.

L'objectif est simple : faire de vous un architecte de l'information. Car celui qui maîtrise l'algorithme maîtrise la logique de conception, domine la technologie et, au final, garde le contrôle sur l'IA.

Bienvenue dans le monde de la pensée algorithmique.

CHAPITRE 1 : LES ELEMENTS DE BASE D'UN ALGORITHME

Avant d'écrire un programme, il faut comprendre une chose essentielle : **un ordinateur ne réfléchit pas**. Il exécute simplement des instructions, exactement comme une recette de cuisine. Par exemple, préparer du foufou suit un ordre précis : 1. Chauffer l'eau, 2. Verser la farine, 3. Mélanger, 4. Servir

Si tu changes l'ordre (verser la farine avant de chauffer l'eau), le résultat sera mauvais. C'est exactement ça, un algorithme : **un ensemble d'instructions ordonnées permettant de résoudre un problème ou d'accomplir une tâche**. Pour bien comprendre comment cela fonctionne, il faut d'abord connaître les éléments fondamentaux qui le composent, qui servent de "briques" de construction dans tous les programmes informatiques.

RÈGLE D'OR DE L'ALGORITHMIQUE : Un bon algorithme doit être **CLOF** : **C**lair (facile à comprendre), **L**ogique (sans contradictions), **O**rdonné (les étapes ont un sens) et **F**ini (il doit s'arrêter)

1.1. QUELQUES DÉFINITIONS DE CONCEPTS

- **Les données :** Les données représentent les informations brutes que l'algorithme va manipuler. Prenons un exemple : la principale différence entre une *donnée* et une *information* est que les données sont des faits bruts (un chiffre, un mot), tandis que l'information est le résultat de ces données une fois organisées pour avoir du sens. "20" est une donnée. "L'étudiant a 20 ans" est une information.
  Elles peuvent être :
  - **Numériques** (ex : 10, 2.5, 2025)
  - **Alphabétiques** (ex : "Matadi", "RDC")
  - **Alphanumériques** (ex : "A25001USK-ST")
  - **Booléennes** (ex : Vrai / Faux)
  - **Dates** (ex : 15/08/2025)
- **Les variables :** Une variable est comme une boîte dans laquelle on stocke une information temporaire pendant l'exécution d'un programme. L'algorithme y place une valeur qui peut changer avec le temps. *Exemple :* Si tu veux mémoriser l'âge d'un étudiant : âge = 20. Ici, "âge" est la boîte. 20 est la valeur stockée à l'intérieur.
- **Les constantes** : Une constante est exactement comme une variable, à une grande différence près : **sa valeur ne change jamais** au cours de l'exécution de l'algorithme. C'est une boîte scellée ! (Ex : la valeur de Pi, ou le nom de l'université, etc.).

1.2. LES TYPES DE VARIABLES ET CONSTANTES

Chaque boîte (variable ou constante) possède un "type", qui définit la nature de la donnée qu'elle a le droit de contenir. Tu ne mets pas de l'eau dans une boîte en carton !

Les types les plus courants sont :

- **Entier :** des nombres sans virgule (ex : un stock, une quantité, un âge).
- **Réel :** des nombres avec virgule (ex : un prix, un taux, un salaire).
- **Caractère :** une seule lettre ou symbole (ex : 'A', '%').
- **Chaîne de caractères :** du texte complet (ex : un nom, une adresse email).
- **Booléen :** uniquement Vrai ou Faux.
- **Date / Heure.**

> ⚠ **ERREUR COURANTE**
>
> Confondre une variable de type "Entier" avec une "Chaîne de caractères". Le nombre 20 permet de faire des calculs. Le texte "20" est juste un dessin du chiffre, tu ne peux pas l'additionner !

1.3. LES OPÉRATEURS

Pour manipuler nos boîtes et nos données, nous utilisons des opérateurs. Nous avons donc Les opérateurs ci-après :

- **Opérateurs arithmétiques** : Ils servent à faire des calculs mathématiques classiques : **Addition (+), Soustraction (-), Multiplication (*), Division entière (/), Division décimale (\)** et **Modulo** % (Il s'agit du reste de la division).
- **Opérateurs relationnels** : Ils permettent de comparer deux valeurs pour prendre des décisions. Le résultat de cette comparaison sera toujours un Booléen : **Vrai** ou **Faux**.
  Nous avons donc comme opérateur relationnels :
  - = : égal à
  - <> : différent de
  - < : strictement inférieur à
  - > : strictement supérieur à
  - <= : inférieur ou égal à
  - >= : supérieur ou égal à
- **Opérateurs logiques** : Ils permettent de combiner plusieurs conditions entre elles. Nous avons comme opérateur logique :
  - **ET (AND) :** Toutes les conditions doivent être vraies. (Ex : Il faut avoir plus de 18 ans ET avoir payé son inscription pour accéder au cours.)
  - **OU (OR) :** Au moins une des conditions doit être vraie. (Ex : Pour voyager, il faut présenter un passeport valide OU une carte d'identité valide.)
  - **NON (NOT) :** Inverse la condition. Si c'est vrai, ça devient faux, et inversement.

1.4. LES INSTRUCTIONS DE BASE

Maintenant que nous avons nos données et nos opérateurs, comment faire agir l'algorithme ?

- **L'affectation :** C'est l'action de donner ou de modifier la valeur stockée dans une variable. On utilise le symbole =. *Exemple :* x = 10 signifie que la boîte nommée "x" contient désormais la valeur 10.
- **L'entrée (Lire / Saisir) :** C'est le moment où l'algorithme demande à l'utilisateur de lui fournir une information. *Exemple :* Demander à l'utilisateur de taper son mot de passe au clavier ou de saisir le nom d'un étudiant.
- **La sortie (Écrire / Afficher) :** C'est le résultat ! L'algorithme communique les informations produites après son traitement. *Exemple :* Afficher la somme d'un calcul à l'écran, ou envoyer un message "Enregistrement réussi".

ASTUCE

Toujours écrire et valider ces étapes au brouillon avec un papier et un stylo avant de commencer à taper la moindre ligne de code sur l'ordinateur !

CHAPITRE 2 : L'ALGORIGRAMME & STRUCTURE DE CONTROLE

Un algorithme écrit sous forme de texte peut parfois sembler lourd, abstrait et difficile à suivre pour un débutant. C'est pourquoi les informaticiens utilisent un outil visuel redoutable : **l'algorigramme** (aussi appelé ordinogramme).

Imagine que c'est le plan d'architecte ou la carte au trésor de ton programme. C'est un schéma qui montre exactement comment l'ordinateur réfléchit, étape par étape, en utilisant des formes géométriques simples.

Pourquoi l'algorigramme est-il si important ?

- Il permet de voir les erreurs de logique avant même de commencer à programmer.
- Il facilite énormément le travail en équipe.
- Il aide à expliquer ton idée brillante à un client ou à quelqu'un qui ne code pas du tout !

2.1. LES SYMBOLES

Dans l'algorigramme, un symbole représente une étape qu'on peut numéroté par ordre croissant. Parmi les symboles nous citons (la liste n'est pas exhaustive) :

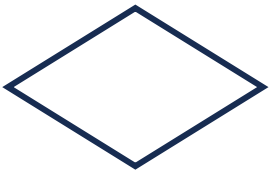
- **Le symbole du début ou de fin de l'algorigramme :** Dans un ordinogramme, également appelé organigramme algorithmique ou algorigramme, la représentation visuelle du déroulement logique d'un programme repose sur des symboles normalisés. Parmi ces symboles, celui du début et de la fin occupe une place fondamentale. Il est généralement représenté par une forme ovale ou un rectangle aux bords arrondis. Ce symbole est destiné à signaler explicitement le point de départ et le point d'arrêt de l'algorithme. Il constitue donc la porte d'entrée et la porte de sortie du traitement décrit dans le schéma.

- **Le symbole de lecture (entrée) ou d'écriture :** Le symbole de lecture, encore appelé symbole d'entrée, est utilisé pour représenter une interaction entre l'algorithme et l'extérieur lorsque l'utilisateur doit fournir des données. Il s'agit généralement d'un **parallélogramme** orienté horizontalement. À l'intérieur, l'instruction de lecture est indiquée, comme par exemple la demande de saisir un nombre ou un texte. Ce symbole marque un point essentiel de communication, car il conditionne souvent la suite des traitements selon les données récupérées. Il garantit ainsi que l'ordinogramme prend en compte la variabilité des entrées fournies par l'utilisateur ou par un dispositif externe.

- **Le symbole d'affichage à l'écran (de sortie) :** Le symbole d'affichage, également appelé symbole de sortie, sert à représenter toute instruction permettant de communiquer des informations vers l'extérieur. Il se présente généralement sous la forme d'un **parallélogramme**, **similaire au symbole de lecture**, mais il est explicitement destiné à montrer le résultat d'un traitement ou un message destiné à l'utilisateur. Ce symbole intervient souvent à la fin d'un processus de calcul ou après une prise de décision pour présenter les données finales, les notifications ou les messages explicatifs. Il met ainsi en évidence la dimension interactive de l'algorithme, en assurant que les traitements effectués conduisent à une sortie visible et exploitable. L'affichage constitue l'aboutissement logique du flux de contrôle, car il valide le travail accompli en rendant le résultat compréhensible pour l'utilisateur.
- **Le symbole de traitement ou action :** Le symbole de traitement représente toute opération effectuée sur les données au cours de l'exécution de l'algorithme. Il s'agit du symbole le plus utilisé dans un ordinogramme. Il est généralement représenté par un **rectangle** à bords droits. À l'intérieur de ce symbole, on inscrit l'action à réaliser, telle qu'une ouverture ou fermeture des fichiers, un calcul, une affectation de valeur à une variable ou encore une manipulation logique. Il constitue le cœur du processus, car il traduit l'évolution des informations en fonction des règles définies. Sans lui, l'ordinogramme ne serait qu'une simple structure de contrôle, dépourvue de traitement effectif. Chaque étape de calcul est ainsi formalisée pour permettre une compréhension claire du cheminement de l'information.

- **Le symbole de décision ou test :** Le symbole de décision correspond aux choix logiques pouvant orienter le déroulement d'un programme vers différentes branches. Il est représenté par un **losange**, à l'intérieur duquel est

inscrite une question portant sur une condition logique, généralement du type vrai/faux, oui/non, 1 ou 0 bref du type booléen. Ce symbole permet de gérer les structures conditionnelles tout en indiquant clairement les différentes directions possibles à la suite du test. C'est grâce à lui que l'algorithme acquiert une dimension dynamique, capable de s'adapter aux situations rencontrées au cours de l'exécution. Les flèches partant du losange montrent donc les différentes issues selon la réponse obtenue.



- **Le symbole de sous-programme :** Le symbole de sous-programme représente l'appel à une procédure ou une fonction déjà définie ailleurs dans l'algorithme. Il est généralement dessiné sous la forme d'un **rectangle** avec deux **barres latérales verticales**. Ce symbole souligne l'importance de la modularité dans la programmation : plutôt que de répéter plusieurs fois le même traitement, il suffit de l'encapsuler dans un sous-programme autonome que l'on pourra réutiliser. En plus de simplifier la conception, l'appel de sous-programme améliore la clarté, la maintenance et la réutilisation du code ou de la logique algorithmique.

- **Les symboles de branchement ou connecteur :** Les symboles de branchement, encore appelé connecteurs, servent à assurer une continuité dans un ordinogramme lorsqu'il est nécessaire de passer d'une partie du schéma à une autre sans confusion. Ils se présentent souvent sous forme d'un **petit cercle ou un carré** contenant une lettre ou un numéro permettant de relier des zones éloignées du diagramme. Ces symboles sont particulièrement utiles dans les ordinogrammes de grande taille, où la lisibilité est essentielle. Grâce au branchement, ils deviennent possibles de maintenir le flux logique même lorsque l'espace ou la mise en page impose des ruptures visuelles.

: Le connecteur Renvoi sur le même support (Papier, Tableau noir etc.)

: Le connecteur Renvoi sur un autre support (Papier, Tableau noir

- **Le symbole de Switch (ou de sélection multiple) :** Le symbole de switch, également connu comme symbole de sélection multiple, est utilisé pour représenter une décision comportant plusieurs choix possibles. Contrairement à la décision binaire classique, ce symbole permet d'orienter l'exécution vers plusieurs sorties selon la valeur d'une variable ou le résultat d'un test complexe. Il peut être représenté par un **losange particulier** ou une **forme dérivée** indiquant clairement la multiplicité des branches. Ce symbole

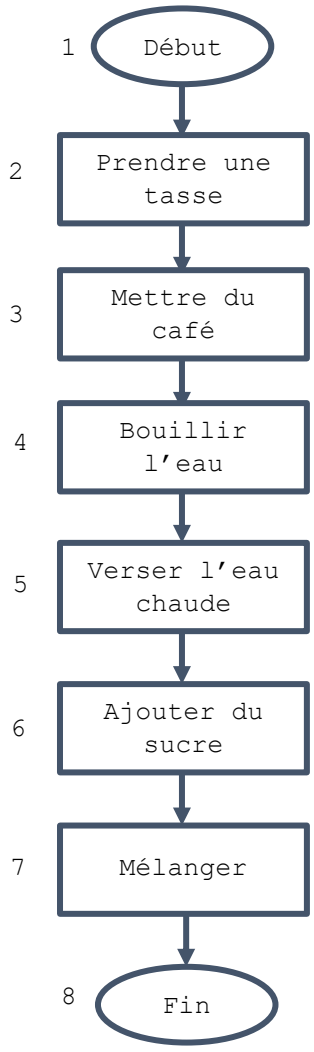
facilite la représentation de structures telles que “selon le cas”, souvent utilisées dans les langages de programmation pour gérer différents scénarios possibles.

2.2. L'ALGORITHMIQUE AU QUOTIDIEN

Prenons un exemple de la vie de tous les jours : préparer du café.

Le matin, tu le fais machinalement. Mais ce que tu fais sans y penser devient un véritable algorithme quand tu l'expliques étape par étape à un robot :

1. Prendre une tasse.
2. Mettre du café.
3. Faire bouillir de l'eau.
4. Verser l'eau chaude.
5. Ajouter du sucre.
6. Mélanger.



En réalité, tu fais déjà de l'algorithmique tous les jours sans le savoir ! L'algorigramme va simplement te permettre de dessiner ce processus.

ASTUCE

Quand tu dessines un algorigramme, utilise toujours les formes standards : un ovale pour marquer le Début et la Fin, un rectangle pour une action (ex: calculer une somme), et un losange pour poser une question (ex: la note est-elle supérieure à 10 ?). Cela rend ton schéma universel et compréhensible par tous les programmeurs du monde.

2.3. LES STRUCTURES DE CONTRÔLE

Maintenant que nous savons dessiner les étapes, comment rendre notre algorithme intelligent ? C'est là qu'interviennent les structures de contrôle.

Sans structure de contrôle, un programme serait comme un robot sans intelligence. Il exécuterait bêtement les instructions de haut en bas et finirait par foncer dans un mur si la situation changeait.

Pour qu'un programme puisse s'adapter et prendre des décisions, il s'appuie sur trois piliers fondamentaux :

2.3.1. La Séquence (L'ordre logique)

C'est la structure la plus basique. Les instructions s'exécutent ligne par ligne, l'une à la suite de l'autre. Exactement comme notre recette de café. Si tu inverses deux lignes, le résultat final est faussé.

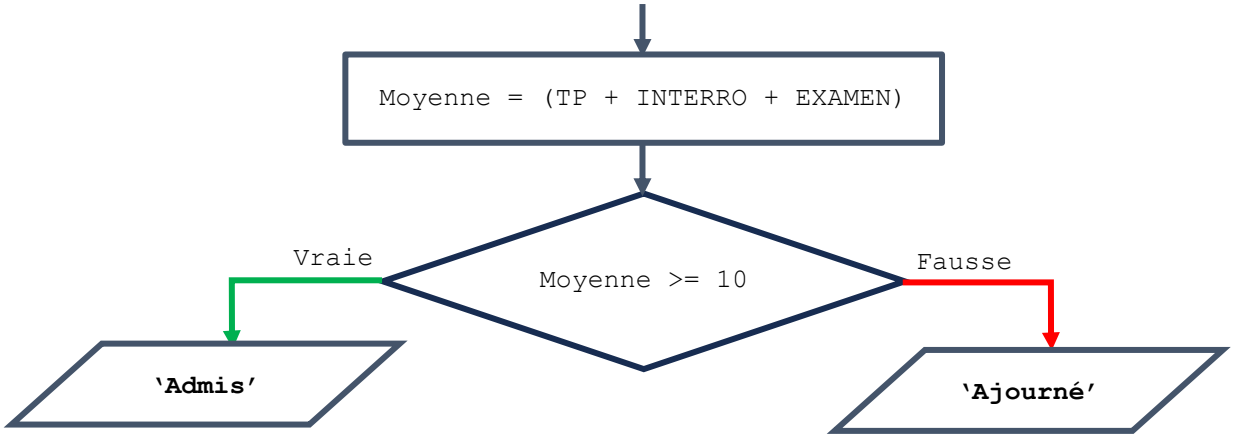
2.3.2. La Condition (Le choix)

C'est le fameux carrefour. Le programme se pose une question et choisit un chemin en fonction de la réponse. On l'appelle souvent la structure **Si... Alors... Sinon...**.

Prenons un exemple concret : Imagine que tu crées un petit programme de délibération pour gérer les cotes des étudiants (comme dans le système LMD) dans un cours donné. Le programme doit décider du sort de l'étudiant.

- **SI** (la moyenne de l'étudiant est supérieure ou égale à 10) **ALORS**
  - Afficher le message : "Étudiant Admis"
- **SINON**
  - Afficher le message : "Étudiant Ajourné"

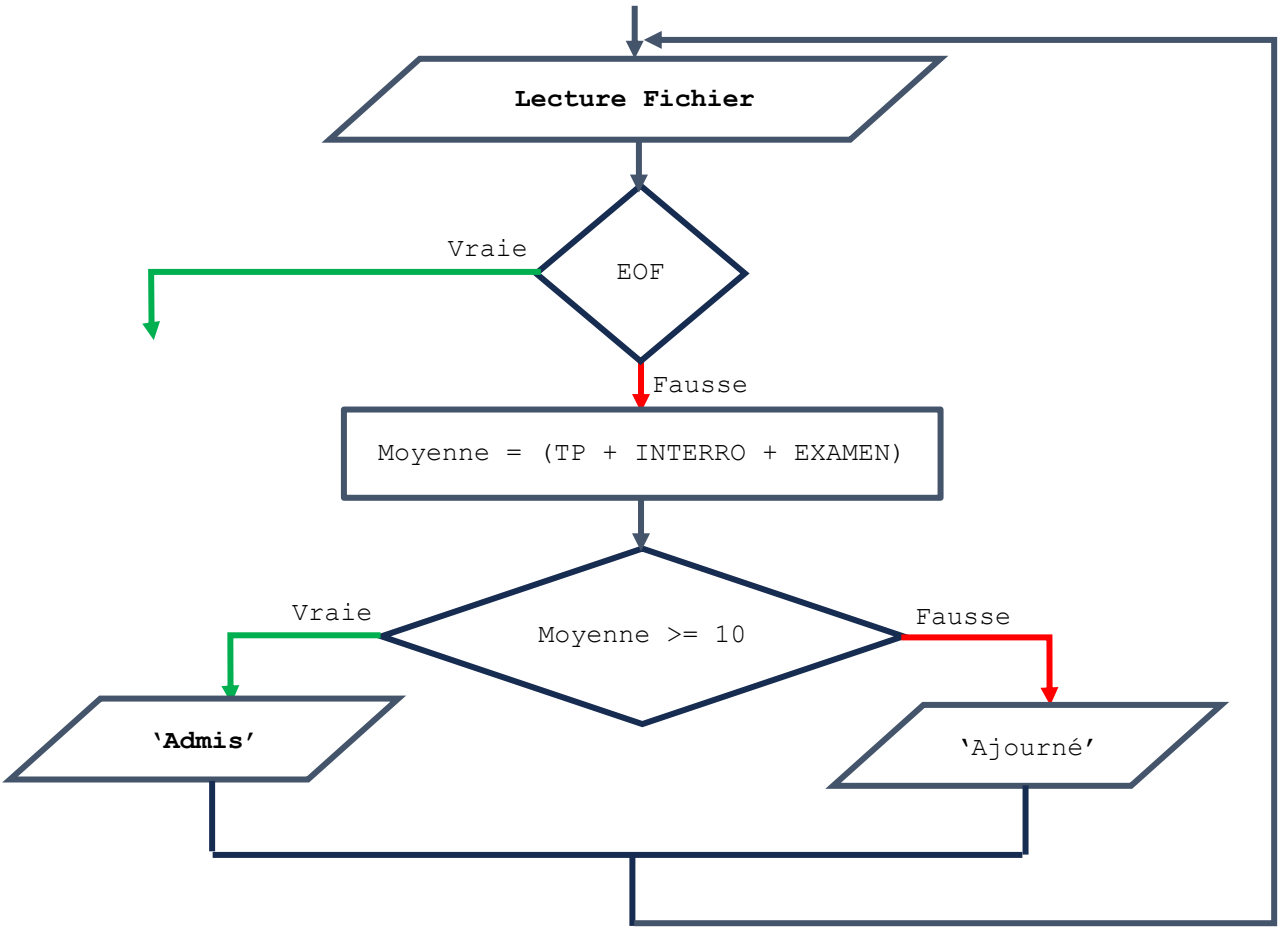
Ici, le programme s'adapte à la donnée qu'il reçoit !



2.3.3. L'Itération ou la Boucle (La répétition)

Imagine que tu dois calculer et vérifier la moyenne non pas d'un seul étudiant, mais de 500 étudiants dans un auditoire. Tu ne vas tout de même pas écrire 500 fois le même bloc d'instructions !

La boucle permet de répéter une action tant qu'une condition est remplie, ou pour un nombre défini de fois.



⚠ **ERREUR COURANTE**

Créer ce qu'on appelle une "boucle infinie". Si tu oublies de dire à ton programme quand il doit s'arrêter (par exemple, "arrête-toi quand tu as traité le 500ème étudiant"), il va tourner en rond pour l'éternité et faire planter l'ordinateur ! N'oublie jamais la règle d'or du Chapitre 1 : **un algorithme doit avoir une fin**. Pour notre illustration, le test EOF (End Of File) permet à notre système de vérifier s'il est arrivé à la fin du fichier contenant les notes 500 étudiants. Si cette condition renvoie la valeur vraie alors notre algorithme doit prendre fin. Si la condition renvoie la valeur fausse cela veut dire qu'il a encore des notes à traiter.

CONCLUSION

Ce chapitre a permis d'approfondir la compréhension des outils fondamentaux de la programmation. Les algorigrammes, en tant que représentations graphiques des algorithmes, offrent une méthode claire et structurée pour concevoir et visualiser la logique des programmes. Ils facilitent la compréhension, la communication et la vérification des processus avant leur implémentation dans un langage de programmation.

Les structures de contrôle, quant à elles, constituent le cœur de la logique conditionnelle et itérative dans la programmation. Elles permettent d'exécuter des blocs de code en fonction de conditions spécifiques ou de répéter des actions jusqu'à ce qu'une certaine condition soit remplie. Les principales structures abordées, telles que les conditions (if, else, else if), les boucles (for, while, do-while), ainsi que les structures de branchement, offrent une flexibilité essentielle pour la résolution de problèmes complexes.

Ce chapitre a également souligné l'importance de maîtriser ces concepts pour développer des programmes efficaces, lisibles et maintenables. La capacité à concevoir des algorigrammes précis et à utiliser judicieusement les structures de contrôle constitue une étape cruciale dans l'apprentissage de la programmation, notamment pour les étudiants qui débutent dans ce domaine. En somme, la maîtrise de ces outils permet de poser une base solide pour la conception algorithmique et la résolution de problèmes informatiques.

CHAPITRE 3 : METHODOLOGIE D'ANALYSE D'UN PROBLEME

Beaucoup d'étudiants échouent en algorithmique non pas parce qu'ils ne savent pas coder… mais parce qu'ils ne comprennent pas le problème.

Avant d'écrire le moindre pseudo-code ou de dessiner un seul schéma, il faut apprendre à réfléchir. Une bonne analyse garantit non seulement un algorithme correct, mais également une solution optimisée, cohérente et facile à maintenir.

Pour analyser un problème efficacement sans jamais te perdre, poses-toi toujours ces 5 questions clés. C'est ce que nous appellerons la **Méthode Etomatique** :

1. **Qu'est-ce qu'on me donne ? (Entrées)** : Quelles sont les données de départ fournies par l'utilisateur ou le système ? ici les données de départ peuvent provenir d'un fichier, d'un autre programme ou saisie avec le clavier par l'utilisateur
2. **Qu'est-ce que je dois trouver ? (Sorties)** : Quel est le résultat final attendu ?
3. **Quelles opérations faire ? (Traitements)** : Quels calculs mathématiques ou fusions de texte sont nécessaires ?
4. **Quelles conditions vérifier ? (Contraintes)** : Y a-t-il des limites logiques (ex : une note ne peut pas être négative) ou des choix à faire ?
5. **Est-ce qu'il faut répéter ? (Boucles)** : Le traitement s'applique-t-il à un seul élément ou à un fichier ?

⚠ **ERREUR COURANTE**

Se précipiter sur le clavier pour écrire du code sans avoir identifié les entrées et les sorties. C'est comme commencer à construire une maison sans avoir fait les plans !

3.1. EXEMPLE 1 : LE CALCUL DE LA MOYENNE (CAS D'UN FICHIER EN ENTREE)

Mettons la Méthode Etomatique en pratique. Imagine que tu développes un petit programme de délibération académique conçu pour gérer la moyenne des notes obtenus dans le cadre du système LMD dans un cours donné. Le problème de base à résoudre pour chaque étudiant est le suivant :

Énoncé : L'université du Savoir Kongo souhaite avoir la moyenne obtenue par chaque étudiant dans le cours de Mathématiques appliquées à l'Informatique. Pour ce faire, il dispose d'un fichier dont la structure se présente comme suite :

MATRICULE	NOMS	TP	INTERRO	EXAM
A025025USK-ST	WATEKO	5	2	7
A025026USK-ST	KIMPAVITA	2	3	8
ETC.	ETC.	ETC	ETC	ETC

On vous demande de mettre en place un algorithme permettant de calculer la moyenne d'un étudiant à partir de trois notes, et d'afficher sa décision (Admis ou Ajourné).

Appliquons nos 5 questions :

1. **Qu'est-ce qu'on me donne ? (Les éléments en entrés) :** Un fichier contenant : Le Matricule (MAT), Le Noms (NOM), Les notes de Travaux Pratiques sur 5 (TP), de l'Interrogation sur 5 (INTERRO) et de l'Examen sur 10 (EXAM).
2. **Qu'est-ce que je dois trouver ? (Les éléments en sortis)** La moyenne finale et la décision d'admission.
3. **Quelles opérations faire ? (Les éléments à calculer)** Additionner les trois notes et diviser le tout par trois : Moyenne = (TP + INTERRO + EXAM) / 3.
4. **Quelles conditions vérifier ?** Vérifier si la moyenne est supérieure ou égale à la note de passage 10/20.
5. **Est-ce qu'il faut répéter ?** Dans cet exercice précis, Oui (on le système doit parcours tous les enregistrements du fichier en entré).

Ici, le problème est simple. Nous avons donc :

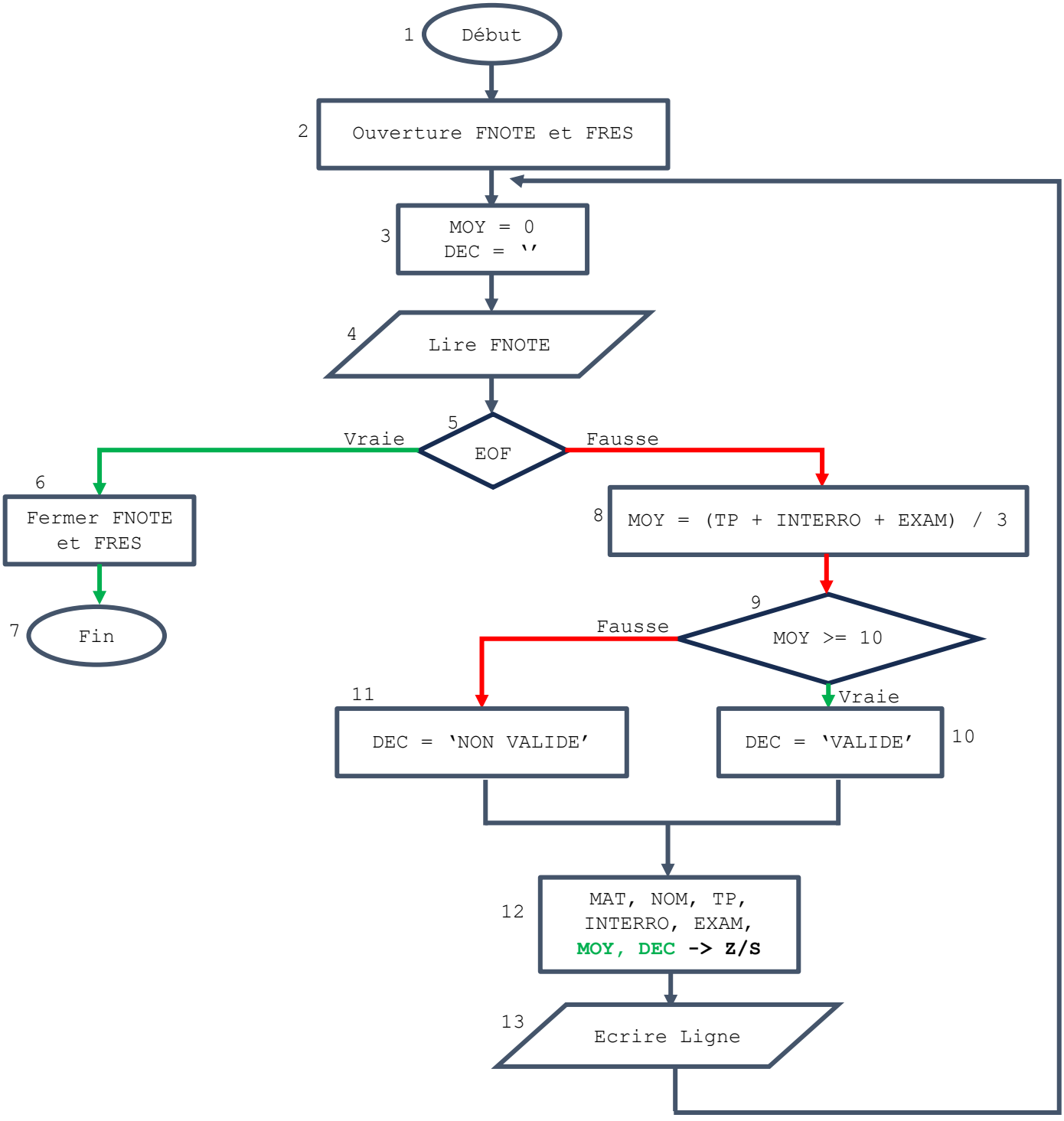
- Une **séquence répétitive** (le calcul mathématique de la moyenne de chaque étudiant).
- Une **condition** (la décision d'admission basée sur le résultat).

ASTUCE

Toujours écrire ces étapes au brouillon, sous forme de liste numérotée, avant de dessiner ton algorigramme ou de coder. Cela libère ton esprit de la syntaxe informatique pour te concentrer à 100% sur la logique !

ALGORIGRAMME

Une fois l'analyse faite, la traduction en Algorigramme devient un jeu d'enfant.



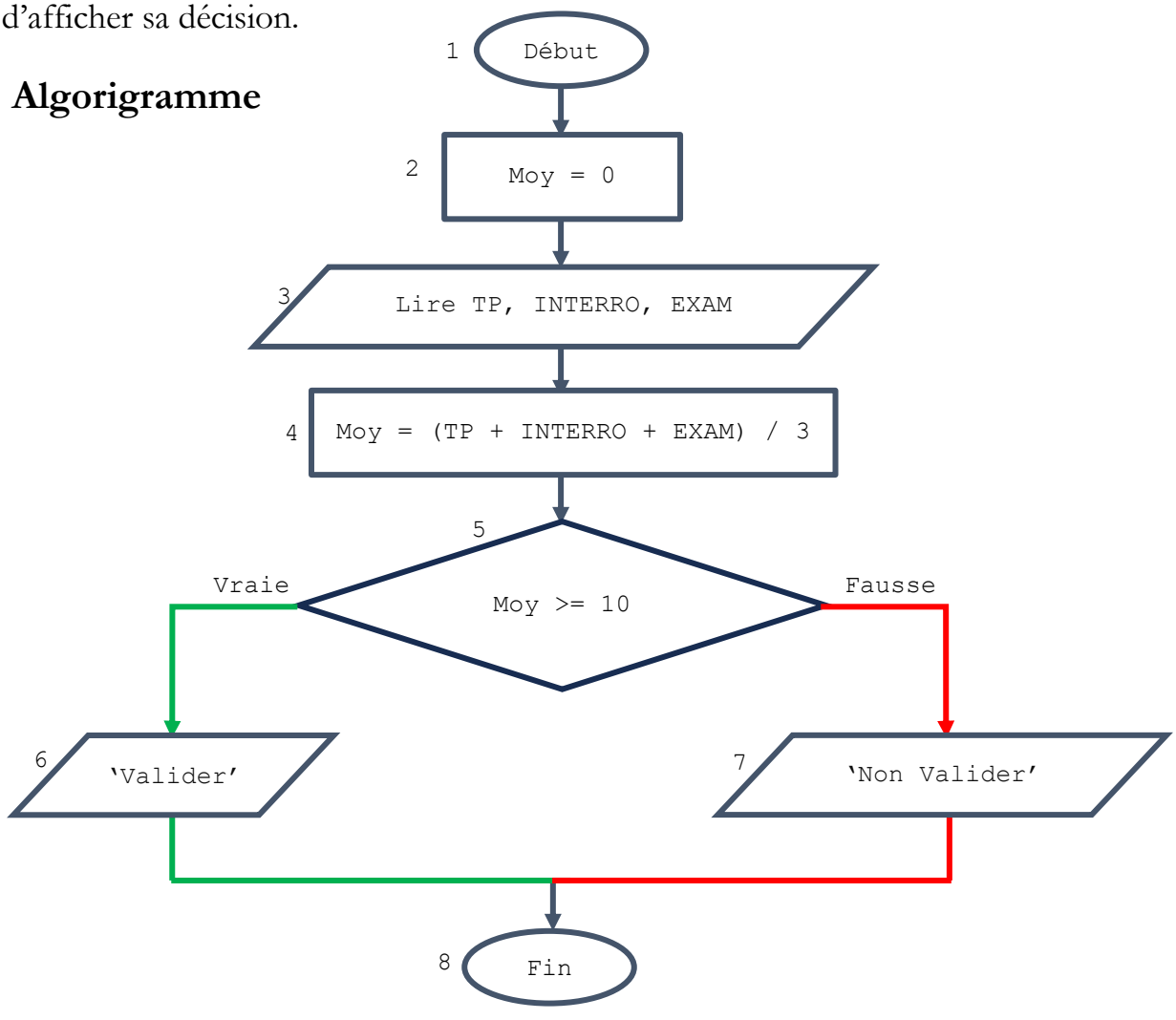
Le Pseudo-code final :

```
1. Début
2. Ouverture de FNOTE et FRES
3. Initialisation MOY = 0 et DEC = ''
4. Lire FNOTE
5. Si EOF Alors
      Aller à 6
   Sinon
      Aller à 8
   FinSi
6. Fermeture de FNOTE et FRES
7. Fin
8. Moy = (TD + TP + Interro) / 3
9. Si Moy >= 10 alors
      Aller à 10
   Sinon
      Aller à 11
   Fin si
10. DEC = 'VALIDE'
11. DEC = 'NON VALIDE'
12. Transfert MAT, NOM, TP, INTERRO, EXAM, MOY, DEC -> Z/S
13. Ecrire ligne, puis aller à 3
```

3.3 EXEMPLE 2 (CAS DE LA SAISIE SUR CLAVIER)

Énoncé : Écrire un pseudo-code permettant de calculer la moyenne d'un étudiant et d'afficher sa décision.

Algorigramme



Algorithme (Pseudo code)

```
1. Début
2. Moy = 0
3. Lire TP, INTERRO, EXAM
4. Calculer Moy = (TP + INTERRO + EXAM) / 3
5. Si Moy >= 10 alors
       Aller à 6
    Sinon
       Aller à 7
6. Afficher 'VALIDE'
7. Afficher 'NON VALIDE'
8. Fin
```

CONCLUSION

La méthodologie d'analyse constitue une étape essentielle dans la résolution de tout problème algorithmique. Elle permet aux étudiants de développer une approche structurée et rigoureuse, facilitant ainsi la compréhension et la résolution de problèmes complexes. Maîtriser cette démarche est crucial pour aborder efficacement chaque étape du processus, depuis la compréhension initiale jusqu'à la conception de solutions concrètes.

Ce chapitre sert de pont entre les notions théoriques et leur application pratique. En comprenant les principes fondamentaux de l'analyse, les étudiants sont mieux préparés à élaborer des algorigrammes et à rédiger des algorithmes précis et efficaces. La capacité à analyser un problème en profondeur permet également d'anticiper d'éventuelles difficultés et d'adopter des stratégies adaptées pour les surmonter.

En résumé, la méthodologie d'analyse n'est pas seulement une étape technique, mais une compétence essentielle qui renforce la logique, la rigueur et la créativité dans la résolution de problèmes. Elle constitue la base sur laquelle repose tout développement algorithmique réussi, et sa maîtrise est indispensable pour tout étudiant souhaitant exceller dans ce domaine.

SECTION EXERCICES : À VOUS DE JOUER !

Maintenant que vous comprenez comment un ordinateur lit et traite l'information, il est temps de passer à la pratique.

> **Rappel :** Avant de tracer votre algorigramme ou d'écrire votre algorithme, prenez toujours un brouillon et appliquez les 5 questions de la **Méthode Etomatique**.

PARTIE 1 : Traitement à partir d'un fichier en entrée

Dans ces exercices, les données sont déjà stockées. Votre algorithme doit lire le fichier ligne par ligne jusqu'à la fin pour produire un résultat global.

Exercice 1 : Statistiques des étudiants (UNISAK)

L'Université du Savoir Kongo (UNISAK), à travers le service des affaires académiques, souhaite connaitre le nombre exact de ses étudiants de sexe masculin et ceux de sexe féminin. Pour ce faire, elle dispose d'un fichier nommé ETUDIANT dont la structure se présente comme suit : MATRICULE, NOM, POSTNOM, PRENOM, SEXE, DATENAISSANCE.

Travail demandé : En tant qu'analyste-programmeur, concevez un algorigramme pouvant résoudre ce problème et afficher les deux totaux à la fin du traitement.

Exercice 2 : Gestion des stocks et réapprovisionnement

Le gérant de l'Établissement ALFREDIVINE souhaite identifier rapidement les produits de son magasin qui sont en rupture de stock afin de passer une commande fournisseur. Il vous remet un fichier STOCK contenant les informations suivantes pour chaque article : CODE_PRODUIT, DESIGNATION, QTE_DISPONIBLE, QTE_SEUIL_MINIMUM.

Travail demandé : Concevez l'algorigramme d'un programme qui lit ce fichier et affiche, pour chaque produit dont la quantité disponible est strictement inférieure au seuil minimum.

Exercice 3 : Délibération annuelle (Système LMD)

À la fin de l'année académique, le jury doit déterminer combien d'étudiants ont validé leur année de Licence. Le système fournit un fichier RESULTATS_LMD structuré ainsi : MATRICULE, NOMS, MOYENNE_SEMESTRE_1, MOYENNE_SEMESTRE_2. Un étudiant valide son année si la moyenne générale des deux semestres est supérieure ou égale à 10/20.

Travail demandé : Écrivez l'algorithme (pseudo-code) qui parcourt ce fichier, calcule la moyenne annuelle de chaque étudiant, et compte le nombre total d'étudiants admis.

PARTIE 2 : Traitement interactif avec saisie utilisateur

Dans ces exercices, l'ordinateur dialogue avec l'utilisateur. Les données sont saisies au clavier en temps réel.

Exercice 4 : La caisse enregistreuse simple

Un caissier a besoin d'un petit programme pour facturer un client. L'algorithme doit demander au caissier de saisir le prix unitaire d'un article et la quantité achetée. Le programme calcule le montant total. Ensuite, l'ordinateur demande si le client a une carte de fidélité (réponse par "O" pour Oui ou "N" pour Non). Si la réponse est Oui, une réduction de 10% est appliquée sur le total.

Travail demandé : Tracez l'algorigramme de ce processus de facturation en n'oubliant pas d'afficher le montant final à payer.

Exercice 5 : Contrôle de saisie d'un mot de passe

Pour sécuriser l'accès au logiciel de gestion d'une entreprise, l'utilisateur doit saisir un code PIN numérique. Le code secret enregistré dans le système est 2026. L'algorithme doit demander à l'utilisateur de saisir son code. S'il se trompe, le programme affiche *"Code incorrect, veuillez réessayer"* et lui redemande de saisir le code, jusqu'à ce que la saisie soit correcte. Une fois le bon code saisi, il affiche *"Accès autorisé"*.

Travail demandé : Concevez cet algorithme en utilisant une structure de contrôle itérative (une boucle).

Exercice 6 : Calculateur d'IMC (Indice de Masse Corporelle)

Un centre médical souhaite un programme rapide pour évaluer la santé de ses patients. Le programme demande à l'infirmier de saisir le poids du patient (en kg) et sa taille (en mètres). L'algorithme calcule l'IMC avec la formule : IMC = Poids / (Taille * Taille).

Si l'IMC est inférieur à 18.5, afficher *"Insuffisance pondérale"*. S'il est compris entre 18.5 et 24.9, afficher *"Poids normal"*. S'il est de 25 ou plus, afficher *"Surpoids"*.

Travail demandé : Élaborez l'algorigramme de cette prise de décision médicale.

A PROPOS DE L'AUTEUR

Alexandre ETOMASANGO OWANGA est enseignant-chercheur à l'Université du Savoir Kongo et développeur informatique spécialisé dans la conception des systèmes d'information.

Il a effectué ses études supérieures à l'Institut Supérieur de Commerce de Matadi, où il a obtenu une Licence (BAC+5) en Gestion Informatique, option Conception des Systèmes d'Information, selon l'ancien système dit PADEM. Ce parcours lui confère le titre d'ingénieur depuis 2023.

Il est à l'origine de plusieurs solutions informatiques, notamment des logiciels personnalisés de gestion académique 360°, déployés dans différents établissements d'enseignement supérieur de la ville de Matadi.

Passionné par la transmission du savoir, il s'engage à rendre l'apprentissage de la programmation simple, accessible et pratique pour tous. À travers ses contenus et ses projets, il œuvre à former une nouvelle génération capable de maîtriser les outils numériques et de développer une pensée structurée et innovante.

A PROPOS DU LIVRE

Et si programmer devenait une manière de penser ?

Apprendre à programmer ne consiste pas seulement à écrire du code. C'est avant tout apprendre à réfléchir, analyser et résoudre des problèmes de manière logique et structurée.

A travers ce livre, *Pensez comme un programmeur*, vous découvrirez les bases essentielles de l'algorithmiques expliquées de façon simple, progressive et accessible à tous. Que vous soyez étudiant, autodidacte ou simplement curieux, cet ouvrage vous guide pas à pas pur développer une nouvelle façon de penser, utilisée par les meilleurs développeurs.

Avec des exemples concrets, des explications claires et des exercices pratiques, vous serez capable de deux choses : Analyser un problème et écrire un algorithme simple pouvant le résoudra.

Ce livre est spécialement conçu pour les débutants, avec une approche pédagogique adaptée aux réalités quotidiennes des apprenants.